09/752,837

| L Number | Hits | Search Text | DB | Time stamp |
|---|---|---|---|---|
| 1 | 18361 | (defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4) | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:32 |
| 3 | 1671 | ((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$) | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:23 |
| 4 | 79431 | operat$4 adj state$ | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:25 |
| 5 | 3631 | time-series | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:26 |
| 7 | 0 | ((((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) with (operat$4 adj state$)) and time-series | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:27 |
| 6 | 4 | (((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) with (operat$4 adj state$) | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:30 |
| 8 | 0 | (((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) and (operat$4 adj state$) and time-series | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:30 |
| 9 | 57 | (((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) with cause | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:32 |
| 10 | 3 | (solution or resolv$4 or solv$4 or answer$4) with ((((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) with cause) | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:42 |
| 11 | 5 | (solution or resolv$4 or solv$4 or answer$4) same ((((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) with cause) | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:43 |

|  |  |  |  |  |
|---|---|---|---|---|
| 12 | 25 | (solution or resolv$4 or solv$4 or answer$4) and ((((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) with cause) | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:43 |
| 13 | 1 |  | USPAT | 2003/08/26 16:45 |
| 14 | 1 |  | USPAT | 2003/08/26 16:45 |
| 15 | 1 |  | USPAT | 2003/08/26 16:45 |
| 16 | 1 |  | USPAT | 2003/08/26 16:46 |
| 17 | 1 |  | USPAT | 2003/08/26 16:47 |
| 18 | 1 |  | USPAT | 2003/08/26 16:47 |
| 19 | 0 |  | USPAT | 2003/08/26 16:48 |
| 20 | 1 |  | USPAT | 2003/08/26 16:48 |

| L Number | Hits | Search Text | DB | Time stamp |
|---|---|---|---|---|
| 1 | 18361 | (defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4) | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:32 |
| 3 | 1671 | ((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$) | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:23 |
| 4 | 79431 | operat$4 adj state$ | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:25 |
| 5 | 3631 | time-series | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:26 |
| 7 | 0 | ((((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) with (operat$4 adj state$)) and time-series | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:27 |
| 6 | 4 | (((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) with (operat$4 adj state$) | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:30 |
| 8 | 0 | (((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) and (operat$4 adj state$) and time-series | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:30 |
| 9 | 57 | (((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) with cause | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:32 |
| 10 | 3 | (solution or resolv$4 or solv$4 or answer$4) with ((((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) with cause) | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:42 |
| 11 | 5 | (solution or resolv$4 or solv$4 or answer$4) same ((((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) with cause) | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:43 |

| | | | | |
|---|---|---|---|---|
| 12 | 25 | (solution or resolv$4 or solv$4 or answer$4) and ((((defect$4 or abnormal$4 or debug$4) adj5 (analys$4 or analyz$4 or test$4)) with (program$ or application$)) with cause) | USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB | 2003/08/26 16:43 |
| 13 | 1 | | USPAT | 2003/08/26 16:45 |
| 14 | 1 | | USPAT | 2003/08/26 16:45 |
| 15 | 1 | | USPAT | 2003/08/26 16:45 |
| 16 | 1 | | USPAT | 2003/08/26 16:46 |
| 17 | 1 | | USPAT | 2003/08/26 16:47 |
| 18 | 1 | | USPAT | 2003/08/26 16:47 |
| 19 | 0 | | USPAT | 2003/08/26 16:48 |
| 20 | 1 | | USPAT | 2003/08/26 16:48 |

PAT-NO: JP411184071A

DOCUMENT-IDENTIFIER: JP 11184071 A

TITLE: PHOTOMASK DEFECT ANALYSIS APPARATUS AND DEFECT ANALYSIS METHOD AS WELL AS RECORD MEDIUM RECORDED WITH PHOTOMASK DEFECT ANALYSIS PROGRAM

PUBN-DATE: July 9, 1999

✓✓

INVENTOR-INFORMATION:

| NAME | COUNTRY |
|---|---|
| FUKUSHIMA, YUICHI | N/A |

ASSIGNEE-INFORMATION:

| NAME | COUNTRY |
|---|---|
| TOPPAN PRINTING CO LTD | N/A |

APPL-NO: JP09353739

APPL-DATE: December 22, 1997

INT-CL (IPC): G03F001/08, G01N021/88 , G06T007/00

ABSTRACT:

PROBLEM TO BE **SOLVED**: To provide a photomask pattern **defect analysis** apparatus and method capable of analyzing the **cause** for the occurrence of abnormality, etc., in a production process by accumulating the detailed information of defects as well as a recording medium recorded with a photomask **defect analysis program**.

SOLUTION: The image data of the defective part detected by a defect observation inspection section 1 is converted to bit map image data in an image input section 2 and thereafter, the defective part is recognized in an image processing section 3. The area, shape, size, kind, etc., of the recognized defective part are decided in a defect deciding section 4. The result of the decision is accumulated as the defect information in a defect information data base 5. The characteristics, cause for the occurrence, etc., of the detected defective part are analyzed in a data analysis section 6 in accordance with the

defect information obtd. by the defect deciding section 4 and the defect
information accumulated in the defect information data base 5. The result of
the analysis is outputted from a data output section 7.

US-PAT-NO: 6567924

DOCUMENT-IDENTIFIER: US 6567924 B1

TITLE: Technique for practically measuring ✓
cycle-by-cycle
repeatable system behavior

---------- KWIC ---------

Brief Summary Text - BSTX (2):
In the field of computer system testing, it is generally desirable to
develop tools for reliably identifying defects in computer systems pertaining
to both hardware and software operation. One prior art approach involves
generating pseudo-random code to run on the system being tested and comparing
emulated results for this test code with results actually obtained on the
computer system being tested. Where there is a discrepancy between emulated
and actual results, the test program generally flags the existence of a
possible defect as a consequence of the discrepancy. However, the discrepancy
between the emulated and actual results and any other information collected
during the first failure may be insufficient to identify the cause of the
suspected defect. More information may need to be collected to narrow the
search for the **cause of the defect, which can be accomplished only if the test
program** can repeat the defective behavior. If the defective behavior cannot be
repeated, it may be very difficult to determine the cause of the defect.

Detailed Description Text - DETX (12):
In the exemplary embodiment of FIG. 1, events which may be counted during a
given timing window to help ensure that a repeatable counter state practically
assures repeatable system behavior, and counters for preserving such
measurements include: 1) the number of clock cycles recorded in counter 110, 2)
the number of transactions issued on the central buses recorded in counter 111,
3) the number of data cycles issued on the central buses recorded in counter
112, 4) the number of delay cycles between winning bus arbitration and issuing

a transaction on the central buses recorded in counter 113, 5) the number of delay cycles needed to **<u>resolve</u>** coherency for transactions on the central buses recorded in counter 114, 6) the number of transactions in progress accumulated each clock cycle on the central buses recorded in counter 115, 7) the number of coherent transactions in progress accumulated each clock cycle recorded in counter 116, and 8) the number of memory page open transactions issued to memory storage recorded in counter 117. It will be appreciated that the count for the events listed above may increase by more than one count in a particular sampling operation. For example, in the case of event type 6 above, if five transactions are in progress when a sampling operation for event type 6 is conducted, the value stored in the counter for event type 6 will be incremented by five rather than by one.

US-PAT-NO: 6304982

DOCUMENT-IDENTIFIER: US 6304982 B1

TITLE: Network distributed automated testing system

---------- KWIC ---------

Brief Summary Text - BSTX (8):
Thus, there is a need in the art for techniques that increase testing
efficiency by **solving** these problems. The present invention **solves** these
problems using a server computer as a central repository for all tests and
results, wherein the server computer is connected to any number of client
computers that perform the tests.

Detailed Description Text - DETX (4):
The present invention improves testing efficiency by using a server computer
as a central repository for all tests performed by any number of connected
client computers. The server computer also acts a central repository for the
results of these tests returned by the client computers. A test manager
executed by the server computer analyzes the results of the tests performed by
the client computers, and determines which tests have passed or failed during
execution, whether the tests failed because of an application or other error,
and which tests should be re-executed by the same or different client computer.
In this manner, the present invention maximizes the testing efficiency of the
resources used and minimizes the amount of time required of the operator to
confirm failures. The operator is presented with tests grouped according to
their status, i.e., tests that succeeded, tests with errors, tests that may
reveal an **application program defect but are difficult or impossible to reproduce, and tests** that reproducibly **cause an application program** to fail.
For the latter category, the operator has test results for each client
computer, so it is immediately apparent whether a defect is universal for the
application program or specific to a particular type or configuration of client
computer. Further, instead of having to interpret whether individual

tests
were successful or failed, a test grouper analyzes the error messages produced
by the application being tested, so that tests that reveal the same defect can
be grouped together. Moreover, a test reducer is iteratively applied to
selected tests to reduce the test to the smallest subset of the original test
that still reveals the defect.

US-PAT-NO: 5854924

DOCUMENT-IDENTIFIER: US 5854924 A

TITLE: Static debugging tool and method

---------- KWIC ---------

Brief Summary Text - BSTX (4):
Executing the **program** in order to **debug the program requires the development**
**of test** suites designed to exercise the **program** in order to **cause** the various
errors to exhibit themselves so that they can be identified and corrected.
Although this approach works well in many circumstances, it does have several
limitations. First the development of the test suites themselves can be
expensive and time consuming. It may bet extremely difficult to identify and
provide all of the test suites necessary to exercise all of the various
possible flow paths which the program being debugged may take. Second, the
quality of the debugging often relies heavily on the breadth and quality of the
test suites, which at times may be lacking.

Detailed Description Text - DETX (23):
As shown in FIG. 6, initially, determination is made as to whether the
current function has a call to another function in step 241. If there is a
call to another function, step 243 determines if there is a call to .sret1,
.sret2, .sret4, or .sret8. Generally, this step determines if there is a call
to a known function for returning an aggregate. If the **answer** is yes, the
logic proceeds to step 220 of FIG. 5. If in step 241, the current function
does not call another function, or if in step 243 there is not a call to
.sret1, .sret2, .sret4, or .sret8, then the logic proceeds to step 245 which
determines if there is a return to %O7+12 or %L7+12. Generally, this step
determines if the current function returns an aggregate itself. If so, the
aggregate flag for the current function is set to true in step 220 of FIG. 5
and, if not, control is passed to step 222 of FIG. 5 which sets the aggregate

US-PAT-NO: 5615332

DOCUMENT-IDENTIFIER: US 5615332 A

TITLE: Debugging aid apparatus

---------- KWIC ---------

Brief Summary Text - BSTX (18):
As described, in the computer with the debugging aid apparatus of the prior
art, since the system call issued from the application task 6 written in a
high-level language calls the supervisor 71 through the high-level language
interface routine 12, the address that the first return-address fetch means 8
fetches as the system call issuing address is always the address of the
high-level language interface routine 12. In other words, when a system call
is issued from one of the application tasks 6, it is not possible to identify
the application task 6 that issued the system call, by referencing the address
that was fetched by the first return-address fetch means 8 and stored in the
table 9. This **causes** a problem in **debugging, when analyzing** an error caused by
the **application** task 6 during the execution of the OS 7 service or processing.

Detailed Description Text - DETX (15):
To **solve** this problem, the present invention provides a flag FLG (see FIG.
12) to indicate whether the system call has been issued through the high-level
language interface routine 12. When a system call is issued, the value of the
FLG is examined; if the flag is set, it is determined that the system call has
been issued from the high-level language interface routine 12, and the number
of system call parameters is obtained from the type of the system call function
code to obtain the return address to the application task 6. Each system call
type is stored with the number of system call parameters in the form as shown
in FIG. 13 in the table 9. Then, the address of the stack where the return
address to the high-level language interface routine 12, the system call
parameters, and the return address to the application task 6 are pushed

in this
order from the top, is incremented by addresses proportional to the number of
parameters, and the return address to the application task 6 is fetched from
the base of the stack and the fetched address is stored into the table 9. On
the other hand, if the FLG is not set, the system call is issued directly from
the assembly language application task; therefore, the return address to the
application task stored at the address pointed to by the SP is stored into the
table 9.

US-PAT-NO: 5522036

DOCUMENT-IDENTIFIER: US 5522036 A
**See image for Certificate of Correction**

TITLE: Method and apparatus for the automatic analysis of
computer software

---------- KWIC ---------

Brief Summary Text - BSTX (9):
Another method of program debugging and analysis involves the use of a
special program, known as a program debugger which monitors the execution of a
software program and oversees its execution. Such debuggers typically offer
enhanced flexibility and more information to the programmer than possible with
the simple PRINT method, particularly with respect to a user interface
presented to the programmer. Utilizing a debugger, a programer will normally
specify one or more lines of the software program as "breakpoints", or points
within a software program which when encountered suspend the execution of the
program so that the programmer may examine processor registers or program
variables. Breakpoints are not limited to specific lines of code, breakpoints
may be used to interrupt execution when an expression changes value, or when a
expression reaches a value. Despite apparent advantages over the PRINT method,
**program debugging and analysis** using known tools suffer from many of the same
infirmities i.e., they do not automate the understanding of **cause**-effect
relationship between faults and failures (manifestation of fault).

Detailed Description Text - DETX (10):
The synchronization CALL() statements reference the SUM-Interface library
263. The SUM-Source code 247 is compiled by compiler/linker 260 and the
resulting code is linked with SUM-Interface library 263 thereby **resolving** the
CALL() references. The output code produced by the compiler/linker 260
operating on SUM-Source code 247 and SUM-Interface library is then executed by
CPU 241. During the execution of this code, the synchronization statements

embedded into SUM-Source and now compiled/linked, are executed which, in turn
invoke the execution of Analytical Engine 244. The creation of SUM-Source 247,
compiling/linking it with SUM-Interface library and sending synchronization
signals to the SUM-Model is not done in the embodiments utilizing the
SUM-Object code or in embodiments utilizing a language interpreter 268
operating on SUM-Repository 246. In those last two embodiments the SUM-Model
243 controls the execution of the target process.

Detailed Description Text - DETX (18):
Definition F1: A Reduced Logic Condition (LC) is a process element with one
control input and two control outputs. One of the two control outputs are
assigned to Boolean 0 (False) and the other control output is assigned to
Boolean 1 (True) which correspond to a **solution** to the logic condition.

Detailed Description Text - DETX (19):
Definition F1.1: The Low Potential **Solution** of a ogic condition is the "0"
Boolean **solution** (False) to the logic condition.

Detailed Description Text - DETX (20):
Definition F1.2: The High Potential **Solution** of a logic condition is the "1"
Boolean **solution** (True) to the logic condition.

Detailed Description Text - DETX (22):
Definition F3: The Main Branch of a process is the path traversed through a
process from an origin of the main branch (process entry or high potential
**solution** of logic condition) to a process terminal when all of the logic
conditions which comprise that path are passed through their low potential
**solutions**.

Detailed Description Text - DETX (39):
The first step performed by SUM-builder 259 is the construction of the
SUM-Frame which is representative of the target process 250. This step is not
needed if the SUM-Graph is constructed directly from the target process. The
target process is processed by the SUM-Builder through the "Next Lowest
Potential" rule or alternatively, through the complementary "Next Highest
Potential" rule. FIG. 22 is a flow chart depicting the steps

associated with
the analysis of a software process by the next lowest potential rule.
The
analyzer proceeds by beginning its analysis at start block 2200.
Successive
statements of the process are parsed by block 2202 until a logic
condition or
process terminal is met. If a logic condition is met as determined by
block
2204, the low potential **solution** of this logic condition is taken as
dictated
by block 2206. When a rewind or exit terminal is encountered as
determined by
block 2208 and untraversed high potential paths remain in the software
process
2210, the Sum-Builder locates the logic condition that was last passed
through
its low potential **solution** and which high potential path was not yet
taken, and
its high potential path is then taken 2214. The analyzer then proceeds
with
the analysis via block 2202. These steps repeat until all of the high
potential paths through the software process being examined are
traversed by
the analyzer.

Detailed Description Text - DETX (45):
Main branches of the graph are directed vertically down and
represent tracks
by which the target process 250 execution progresses, unless the track
is
switched as the result of the high potential **solution** to a logic
condition
encountered in the track. As shown in FIG. 5(A), every track has a
unique b
coordinate. Every track of the graph is parallel to one another.

Detailed Description Text - DETX (50):
All possible changes in the direction of control is reduced to
one--to the
right as the chart is traversed. In this manner, all high potential
**solutions**
to a logic condition are represented as a shift to the right in the
graph.
Only backward (up and left) jumps of unconditional control statements
are
possible when a software process is represented by a reduced flow
chart. An
exception to this general rule is an exit from a loop. Such an exit
may be
made forward.

Detailed Description Text - DETX (51):
Definition G2: The negative state of a logic condition is the state
of a

logic condition which, when evaluated results in its 0-**solution** or low potential **solution**. With reference to FIG. 5(A), logic condition 1 is represented as L1 at k,b coordinate 2,1. Traversing the SUM-Graph shown in
FIG. 5(A) through L1 with L1 in its negative state would result in relocating
to k,b position 3,1 which is represented by SUM-Object set member /7 in the
graph.

Detailed Description Text - DETX (52):
Definition G3: The positive state of a logic condition is the state of a
logic condition which, when evaluated results in its 1-**solution** or high potential **solution**. With reference to FIG. 5(A), traversing the SUM-Graph
shown in FIG. 5(A) through L1 with L1 in its positive state would result in
relocating to k,b position 3,7 which is represented by SUM-Object set member E1
in the graph.

Detailed Description Text - DETX (53):
Definition G4: The binary address (BA) of an SUM-Graph element shows that
elements position relative to process logic constructs and is represented by
the integers 1 and 0 where the integers comprising the BA represent the **solutions** of the logic conditions in the t-pass up to that element with the
process entry assigned the binary address of 1.

Detailed Description Text - DETX (55):
Referring once again to FIG. 5(A), the binary address of k,b position 8,1
(L3) is 100. This is constructed from the process address (binary address 1),
the negative **solution** (0) to the logic condition L1 located at k,b position 2,1
and a negative **solution** to logic condition L2 located at k,b position 6,1 in
the graph. As a further example the binary address of A7 (located at k,b
position 7,5 in FIG. 5(A)) is 101. This is constructed from the process
address (binary address 1), the negative **solution** (0) to the logic condition L1
located at k,b position 2,1 in the graph and the positive **solution** (1) to the
logic condition L2 located at k,b position 6,1 in the graph.

Detailed Description Text - DETX (58):
FIG. 26 is a flow chart depicting the steps associated with the analysis of

a software process by the next lowest potential rule and the construction of
the resulting SUM-Graph. The analyzer proceeds by beginning its analysis at
start block 2600. The position on the graph is assigned to k,b position of 1,1
2601, and the entry binary address is assigned to 1 as indicated in block 2602.
Successive statements of the process are examined by block 2603 with the
SUM-Graph(k,b) position being populated with the SUM-Object set member
representative of the examined statement as required by block 2604. The
repository is updated in block 2605 and if the statement under examination
presently is a logic condition 2606, then the low potential **solution** to the
logic condition is taken 2609, the present k,b position is updated by k=k+1 as
stated in block 2612 and the statement reexamination block 2603 is reentered.

Detailed Description Text - DETX (61):
If there were untraversed high potential paths as determined by block 2616,
then the SUM-Graph position is updated to the last logic condition passed
through its 0 **solution** (block 2617) and a new main branch of that logic
condition is begun at SUM-Graph position(nextk, nextb) where nextk is assigned
to k(logic condition)+1 and next b is assigned to the last used b+1. The last
used b is updated and the process then continues with the examination statement
2603.

Detailed Description Text - DETX (62):
The complementary to the Next Lowest Potential method (i.e., the Next
Highest Potential method) that was mentioned earlier means the following:
within the Next Highest Potential method, main branches would start not from
the positive **solutions** of logic conditions, but from the negative **solutions** of
logic conditions, and the main branches would pass not through the negative
**solutions** of the logic conditions, but through the positive **solutions** of logic
conditions.

Detailed Description Text - DETX (65):
In the present preferred embodiment, the SUM-Graph is built in only two
directions--down and to the right as one looks at the graph. When the

SUM-Graph so constructed is traversed, the traversal proceeds down the graph
and can only be interrupted by shifting to the right as a result of a high
potential **solution** to an encountered logic condition.

Detailed Description Text - DETX (66):
   Referring once again to FIG. 5(A) the graph traversal would begin at k,b
position 1,1 (process entry) and proceed through logic condition L1 (k,b
position 2,1), through label /7 (k,b position 3,1), through arithmetic A1 k,b
position 4,1) and so on until terminal *1 (k,b position 10,1) is reached. Such
a traversal would proceed in a single direction, down the graph. If however, a
high potential **solution** to a logic condition was passed through during
traversal then a shift to the right would have taken place. For example,
during the above traversal, if the logic condition L1 (k,b position 2,1) were
passed through its high potential **solution** then a shift in flow would take
place to E1 (k,b position 3,7).

Detailed Description Text - DETX (69):
   Referring once again to FIG. 5(A), it becomes obvious that statement A7 (k,b
position 7,5) is not accessible from statement W1 (k,b position 16,4). That is
because there is no possible path from statement W1 that would proceed through
statement A7. Statement A7 is however, accessible from statement X1 (k,b
position 9,1). That is because statement *1 (k,b position 10,1) repositions
the path taken to /1 (k,b position 5,1) which immediately precedes statement L2
(k,b position 6,1). The high potential **solution** of L2 leads directly to
statement A7. Therefore, the analyzer which is the object of the present
invention is able to determine whether statement(s) or elements of target
process 250 could be influenced by the correctness of a particular statement or
statements within that process. When the target process 250 is represented in
SUM-Graph form as shown in FIG. 5(A) and is used subsequently for the
construction of SUM-Model 243, the analyzer is capable of evaluating the
possibility for a statement s1 of target process 250 to be responsible for the
misbehavior of statement s2 of target process 250 by defining accessibility

between s1 and s2 necessary condition for cause-effect relationship. Alternatively, the analyzer is capable of evaluating the potential for statement s2 to be effected by the modification of statement s1.

Detailed Description Text - DETX (72):
The shift potential is the potential of a logic condition to increment the
SUM-Graph b coordinate as a result of a positive **solution** of the logic condition. With reference to FIG. 5(A), logic conditions L3 (k,b position
8,1), L4 (k,b position 11,2), L5 (k,b position 15,3) and L6 (k,b position 8,5)
all have a shift potential of 1 because a positive **solution** to any of them
results in a right shift by one of the b coordinate. Similarly, logic condition L2 (k,b position 6,1) has a shift potential of 4 and logic condition
L1 (k,b position 2,1) has a shift potential of 6.

Detailed Description Text - DETX (76):
Definition G8: The shift potential of LC is the potential of LC to increment
b-coordinate as a result of a positive **solution** of the LC.

Detailed Description Text - DETX (113):
As an example of data dependent endless loop and with reference to FIG. 9,
if there are no elements on the pass between /1 912 and *1 930 that can redefine the values, being examined by L2 914 and L3 924, then terminal *b1 930
will produce a data dependent endless loop as soon as L2 914 and L3 924 will be
**resolved** by their Low Potential **solutions** the first time (since the state of L2
914 and L3 924 will never change during the forward execution of the target
process.)

Detailed Description Text - DETX (114):
As an example of a data independent endless loop and with reference to FIG.
9, a data independent endless loop construction is presented by Main Branches
MB2, MB3, MB4. If L3 924 is ever **solved** positively, the process will go in the
infinite loop, no matter, what data manipulation statements are coded within
those branches.

Detailed Description Text - DETX (116):
Entry /q 922 in FIG. 9 does not correspond to any terminal and is shown only
as an example of the fact, that the position of L3 924 in the negative

subfield
of L2 914 with rewind terminal *b1 rewinding the process control
directly in
front of L2 914 is not necessarily a fault, even if L2 914 and L3 924
are not
data dependent on the path /1 to *1 930. This is, because, if L3 924
had
received control through the entry /q 922, returning the control to the
entry
/i 912 will not necessarily result in L2 914 being **solved** negatively
thus in
creating an endless loop. Data Independent Endless Loops are definite
faults
in the process construction.

Detailed Description Text - DETX (150):
A positive **solution** of a conditional control event results in a
control
shift of a process to the right as viewed within an SUM-Graph or
reduced flow
chart. A negative **solution** to a conditional control event results in
the
direction of a process to proceed downward as viewed within an
SUM-Graph or
reduced flow chart.

Detailed Description Text - DETX (181):
In addition, structures representing L and D elements have an
additional
attribute which represents the increment in the b coordinate when
traversed
through their positive, or high potential **solution**. This corresponds
to the
shift potential previously described with respect to the SUM-Graph.
For
example, element L1 (k,b position 2,1) shown in FIG. 5(A) would have a
shift
potential of 6 recorded in its shift potential attribute field.

PAT-NO: JP02001051864A

DOCUMENT-IDENTIFIER: JP 2001051864 A

TITLE: TEST CONDUCTING SYSTEM FOR DATA PROCESSOR

PUBN-DATE: February 23, 2001

INVENTOR-INFORMATION:

| NAME | COUNTRY |
|---|---|
| IWATA, TAKAYUKI | N/A |
| KODAMA, YUTAKA | N/A |
| MITSUMATA, HIROICHI | N/A |

ASSIGNEE-INFORMATION:

| NAME | COUNTRY |
|---|---|
| HITACHI LTD | N/A |

APPL-NO: JP11228243

APPL-DATE: August 12, 1999

INT-CL (IPC): G06F011/22

ABSTRACT:

PROBLEM TO BE **SOLVED**: To analyze the cause of an error if the execution of a test instruction sequence is stopped due to a processor logic defect or if an infinite loop is entered and no execution result is obtained.

**SOLUTION**: This test conducting system has a test environment setting process 101, an instruction emulator 102 which generates an expected value, a test instruction sequence execution control process 103, a result comparing result 104, and a fault factor specifying process 105, and an instruction and an object function which **cause** a fault are found out to investigate the fault factor if the expected value generated by the instruction simulation of the test instruction sequence becomes discrepant with the execution result of the test instruction sequence on a data testing device or, if the execution of the test instruction sequence is stopped due to the fault resulting from a processor logic defect, or if an infinite loop is entered and no execution

result is obtained, thereby evading the occurrence of a fault caused by the
same **defect factor in the subsequent execution of a test program**.